YAZILIM KALİTE GÜVENCESİ VE TESTİ

Dönem Sonu Projesi

Akademik Yıl: 2025-2026 Güz Dönemi

Projenizi Github'ta barındırmalısınız. Son teslim tarihi geldiği anda projeniz **public repository** olmalıdır.
Son Teslim Tarihi: 17 Ocak 2026 saat 23:59

Ad Soyad: _____

Öğrenci Numarası: _____

PROJE HAKKINDA

Genel Tanım

En az **5 farklı kaynak (resource)** içeren bir REST API oluşturacaksınız. Örnek kaynaklar: `users`, `products`, `orders`, `categories`, `reviews` gibi.

API Gereksinimleri

- Her kaynak için aşağıdaki HTTP metodlarını desteklemelisiniz:
  - **GET**: Hem liste (`/api/users`) hem de tekil kaynak (`/api/users/1`)
  - **POST**: Yeni kaynak oluşturma (`/api/users`)
  - **PATCH/PUT**: Mevcut kaynağı güncelleme (`/api/users/1`)
  - **DELETE**: Kaynak silme (`/api/users/1`)
- En az bir kaynağın diğer kaynaklarla ilişkisi olmalıdır (örn: User-Order ilişkisi)
- Uygun HTTP durum kodları kullanılmalıdır (200, 201, 400, 404, 500)
- Veritabanı kullanımı zorunludur (SQLite, PostgreSQL, MongoDB vb.)
- **API dokümantasyonu zorunludur**: Swagger/OpenAPI standardında API dokümantasyonu sağlanmalıdır
  - Swagger UI veya benzeri bir arayüz üzerinden API'ye erişilebilmeli (örn: `/api-docs`)
  - Tüm endpoint'ler, parametreler, request/response şemaları dokümante edilmeli
  - OpenAPI 3.0+ spesifikasyonuna uygun olmalıdır

Teknoloji Önerileri

- **Backend**: Node.js (Express), Python (Flask/FastAPI), Java (Spring Boot), C# (.NET Core)
- **Test Framework**: Jest, PyTest, JUnit, xUnit, Mocha
- **Veritabanı**: PostgreSQL, MySQL, SQLite, MongoDB
- **API Dokümantasyonu**: Swagger UI, Springdoc OpenAPI, Flask-RESTX, NestJS Swagger, FastAPI (built-in)

Teslim Formatı

- Github repository'niz **public** olmalıdır
- **README.md** dosyası şunları içermelidir:
  - Proje açıklaması ve kullanılan teknolojiler
  - Kurulum talimatları (adım adım)
  - API endpoint'lerinin listesi ve kullanım örnekleri
  - Swagger/OpenAPI dokümantasyonuna erişim URL'si (örn: `http://localhost:3000/api-docs`)
  - Test çalıştırma komutları
  - CI/CD badge'leri (GitHub Actions, Codecov)
- Github bağlantı adresini forma girmelisiniz: `https://forms.gle/PZhigBPLfAhe874c9`

SORULAR VE DEĞERLENDİRME KRİTERLERİ

1. (20 points) **Birim Testler (Unit Tests)**

   Projeniz için kapsamlı birim testler oluşturun.

   Minimum Gereksinimler:

   - En az **15 farklı birim test** yazılmalıdır
   - Test edilmesi gerekenler:
     - İş mantığı fonksiyonları (business logic)
     - Veri doğrulama (validation) fonksiyonları
     - Yardımcı fonksiyonlar (utility functions)
     - Model/Entity metodları
   - **Minimum %60 kod coverage** sağlanmalıdır
   - Her test anlamlı test adına sahip olmalıdır
   - Pozitif ve negatif senaryolar test edilmelidir

   Değerlendirme:

   - Test sayısı ve kalitesi: 8 puan
   - Code coverage oranı: 6 puan
   - Test organizasyonu ve isimlendirme: 6 puan

2. (20 points) **Entegrasyon Testleri (Integration Tests)**

   Sistemin farklı bileşenlerinin birlikte çalışmasını test eden entegrasyon testleri oluşturun.

   Minimum Gereksinimler:

   - En az **10 entegrasyon testi** yazılmalıdır
   - Test edilmesi gerekenler:
     - API endpoint'leri (HTTP request/response)
     - Veritabanı işlemleri (CRUD operasyonları)
     - İlişkili kaynaklar arası işlemler (örn: User-Order)
     - Hata durumları (404, 400, 500)
   - Her HTTP metodu (GET, POST, PATCH, DELETE) test edilmelidir
   - Test veritabanı kullanılmalıdır (test isolation)

   Değerlendirme:

   - API endpoint testlerinin kapsamlılığı: 10 puan
   - Veritabanı entegrasyon testleri: 5 puan
   - Test veri yönetimi (setup/teardown): 5 puan

3. (20 points) **Sistem Testleri (System/E2E Tests)**

   Uygulamanın bir bütün olarak çalışmasını test eden uçtan uca testler oluşturun.

   Minimum Gereksinimler:

   - En az **5 farklı senaryo** test edilmelidir
   - Test senaryoları örnekleri:
     - Kullanıcı kaydı → Giriş → Ürün ekleme → Sipariş oluşturma
     - Ürün listeleme → Detay görüntüleme → Güncelleme → Silme

  - Birden fazla kaynakla ilgili kompleks iş akışları
- Her senaryo gerçek kullanım durumlarını simüle etmelidir
- Testler bağımsız çalışabilmeli (her test kendi verilerini oluşturmalı)

Değerlendirme:

- Senaryo kapsamlılığı ve gerçekçiliği: 10 puan
- Testlerin bağımsızlığı: 5 puan
- Test dokümantasyonu: 5 puan

4. (Bonus points) **CI/CD ve Test Otomasyonu (BONUS)**

   Bu soru bonus puandır ve çarpan sistemi ile çalışır.

   Seviye 1 - GitHub Actions CI (Çarpan: x1.5):

   - Tüm testlerin GitHub Actions ile otomatik çalışması
   - Her push ve pull request'te testlerin tetiklenmesi
   - `.github/workflows/` klasöründe çalışan YAML dosyası
   - Test sonuçlarının action loglarında görülebilmesi

   Seviye 2 - Code Coverage Raporlama (Çarpan: x2.0):

   - Codecov, Coveralls veya benzeri servise entegrasyon
   - Her commit için otomatik coverage raporu yüklenmesi
   - README.md'de coverage badge'inin görünmesi
   - Coverage raporlarının detaylı olması

   Puanlama Hesabı:

   - **Temel puan**: Soru 1 + Soru 2 + Soru 3 (maksimum 60 puan)
   - **Seviye 1 tamamlandıysa**: Temel puan x 1.5= Toplam puan
   - **Seviye 2 tamamlandıysa**: Temel puan x 2.0= Toplam puan
   - **Maksimum puan**: 60 x 2.0= 120 puan

   Örnek Hesaplama:

   - Öğrenci 1-2-3. sorulardan toplam 45 puan aldı
   - Sadece GitHub Actions yaptı: 45 x 1.5= 67.5 puan
   - Hem GitHub Actions hem Codecov yaptı: 45 x 2.0= 90 puan

GENEL DEĞERLENDİRME TABLOSU

Kriter	**Puan**	**Açıklama**
Birim Testler	20	15+ test, %60+ coverage, iyi organizasyon
Entegrasyon Testleri	20	10+ test, tüm endpoint'ler, DB entegrasyonu
Sistem Testleri	20	5+ senaryo, gerçekçi akışlar, bağımsız testler
Temel Toplam	**60**	**Minimum başarı notu için gerekli**
GitHub Actions (Bonus)	x1.5	Tüm testlerin CI'da çalışması
Codecov Entegrasyonu (Bonus)	x2.0	Coverage raporlama + GitHub Actions
Maksimum Toplam	**120**	**Tüm bonuslar ile**

Önemli Notlar

- **Swagger/OpenAPI dokümantasyonu zorunludur**. Dokümantasyon eksik veya hatalı olan projeler değerlendirmeye alınmaz.
- Kod kalitesi önemlidir. Kod standartlarına uygun yazılmalıdır.
- Commit mesajları anlamlı ve düzenli olmalıdır.
- Kopyala-yapıştır tespit edilirse proje geçersiz sayılır.
- Proje çalışmıyorsa veya testler başarısız oluyorsa değerlendirmeye alınmaz.
- REST API endpoint'leri çalışır durumda olmalı ve Swagger UI üzerinden test edilebilmelidir.